University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Lab 0 is due this Friday (not graded)
- Recitations start this week
- Homework 1 will be assigned this Friday
- JDB Example available on Canvas
- Draft slides and handouts available on Canvas
- CourseMIRROR pre-survey and consent form

Today's Agenda

- A technique for modeling runtime of algorithms
  - $\sum_{all\ statements} Cost * frequency$
- Extracting the rate of growth of a function
  - Ignoring lower-order terms and multiplicative constants
  - The Big O family
- Backtracking algorithm
  - Overall (recursive) structure
  - Relationship to the search space tree
- Examples
  - PIN/Password Cracking
  - 8 Queens
  - Boggle game

Let’s consider ThreeSum problem from text

- 3-sum Problem:
  - Given a set of arbitrary integers find out how many **distinct** triples sum to exactly zero
  - do you have questions on the problem specification?
- Example input:
  - 5, -1, 2, -3, -2, 1, 0
  - what should the output be?

5, -1, 2, -3, -2, 1, 0

Brute-force solution

Enumerate all possible distinct triples and check their sums

```
cnt = 0
for each distinct triple
    if sum of triple equals zero
        increment cnt
```

Brute-force solution

```java
public static int count(int[] a) {
    int n = a.length;
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            for (int k = j+1; k < n; k++) {
                if (a[i] + a[j] + a[k] == 0) {
                    cnt++;
                }
            }
        }
    }
    return cnt;
}
```

Mathematically modelling runtime

- Runtime primarily determined by two factors:
  - **Cost** of executing each statement
    - Determined by machine used, environment running on the machine
  - **Frequency** of execution of each statement
    - Determined by program and input

What is the runtime?

A technique for modeling runtime of algorithms

- $\sum_{all\ statements} Cost * frequency$

- Split the algorithm into blocks such that
  - the code statements in each block have the same frequency
- $\sum_{all\ blocks} Cost * frequency$

Algorithm Analysis Example 1

$$\texttt{for}(\overset{1}{\boxed{i=0}};\ \overset{n+1}{\boxed{i<n}};\ \overset{n}{\boxed{i{+}{+})\quad a[i]=i;}}$$

Algorithm Analysis Example 2

1

0 or 1   0 or n

```
if( x >0 ){
    for( i=0 ; i<n ; i++ )
        a[i] = i;
```

2

Algorithm Analysis Example 3

1

$$\text{for}(\boxed{i = n};\ i >= 1;\ i = i/2)$$

$$a[i] = i;$$

$\log n$

What is the runtime?

```java
public static int count(int[] a) {
    int n = a.length;
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            for (int k = j+1; k < n; k++) {
                if (a[i] + a[j] + a[k] == 0) {
                    cnt++;
                }
            }
        }
    }
    return cnt;
}
```

A couple useful Math formulae

$\sum$ arithmetic Series:

$$1 + 2 + 3 + 4 + \cdots + n$$

$$= \#\text{terms}\left(\frac{\text{first term} + \text{last term}}{2}\right)$$

$$= \Theta(\#\text{terms} * \text{largest term})$$

$\sum$ geometric Series:

$$1, 2, 4, 8, 16, \cdots$$

$$1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \cdots$$

$$= \Theta(\text{largest term})$$

Runtime Analysis of 3-loop Algorithm for ThreeSum

1

n

```
for (i=0; i<n; i++)
  for (j=i+1; j<n; j++)
    for (k=j+1; k<n; k++)
      if (a[i]+a[j]+a[k]==0)
        cnt++;
```

n^2 n^3 $0 \ldots n^3$

Best Case: $1+n+n^2+n^3+0=\Theta(n^3)$

Worst Case: $1+n+n^2+n^3+n^3=\Theta(n^3)$

Enter Asymptotic Analysis

Algorithm Analysis

- Determine *resource usage* as a function of *input size*
- Measure ***asymptotic*** performance
  - Performance as input size increases to infinity

Asymptotic performance

Focus on the **order of growth** not on exact values

- How fast the function value increases when the input increases

Common orders of growth

- Constant - 1
- Logarithmic - log n
- Linear - n
- Linearithmic - n log n
- Quadratic - n^2
- Cubic - n^3
- Exponential - 2^n
- Factorial - n!

Side note

What does $log_2 n$ mean?

Quick algorithm analysis

- How can we determine the order of growth of a function?
  - Ignore lower-order terms
  - Ignore multiplicative constants

Example

$$5n^3 + 53n + 7 \ \rightarrow n^3$$

- Can we say $5n^3 + 53n + 7 = n^3$?
- No! We need a mathematical notation
- $5n^3 + 53n + 7 = O(n^3)$
- It means the order of growth of $5n^3 + 53n + 7$ is no more than ($\leq$) the order of growth of n^3

The Big O Family

- O roughly means ≤
  - Big O
- o roughly means <
  - Little O or O-micron
- Ω roughly means ≥
  - Big Omega
- ω roughly means >
  - Little Omega
- ϴ roughly means =
  - Theta
- Relationships are between orders of growth, not between exact values!

Formal Definitions

- May also see:
  - $f(x) \in O(g(x))$ or
  - $f(x) = O(g(x))$
- used to mean that $f(x)$ is $O(g(x))$
- Same for the other functions

Formal definitions

$O(n)$

$10n$ $\quad$ $\log n$

$50n+1$

$3\sqrt{n}+10$ $\quad$ 7

n^2

$10n \in O(n)$

$\log n \in O(n)$

$n^2 \notin O(n)$

Theta vs. Tilde

Tilde approximation (~)

- Same as Theta but keeps constant factors
- Two functions are Tilde of each other is they have the same order of growth and the same constant of the largest term

$$5n = \Theta(5{,}000{,}000{,}000\, n)$$

$$\not\sim 5{,}000{,}000{,}000\, n$$

Wait…

- Assuming that definition…
  - Is ThreeSum $O(n^4)$?
  - What about $O(n^5)$?
  - What about $O(3^n)$??
- If all of these are true, why was $O(n^3)$ what we jumped to to start?

A better algorithm for ThreeSum

- What if we sorted the array first?
  - Pick two numbers, then binary search for the third one that will make a sum of zero
    - a[i] = 10, a[j] = -7, binary search for -3
    - Still have two for loops, but we replace the third with a binary search
      - Runtime now?
    - What if the input data isn't sorted?
- What about the sorting time?

$$\underbrace{n^2}_{\text{all pairs}}\ \underbrace{\log n}_{\text{Binary Search}} + \underbrace{\cancel{n \log n}}_{\text{Sorting}}$$

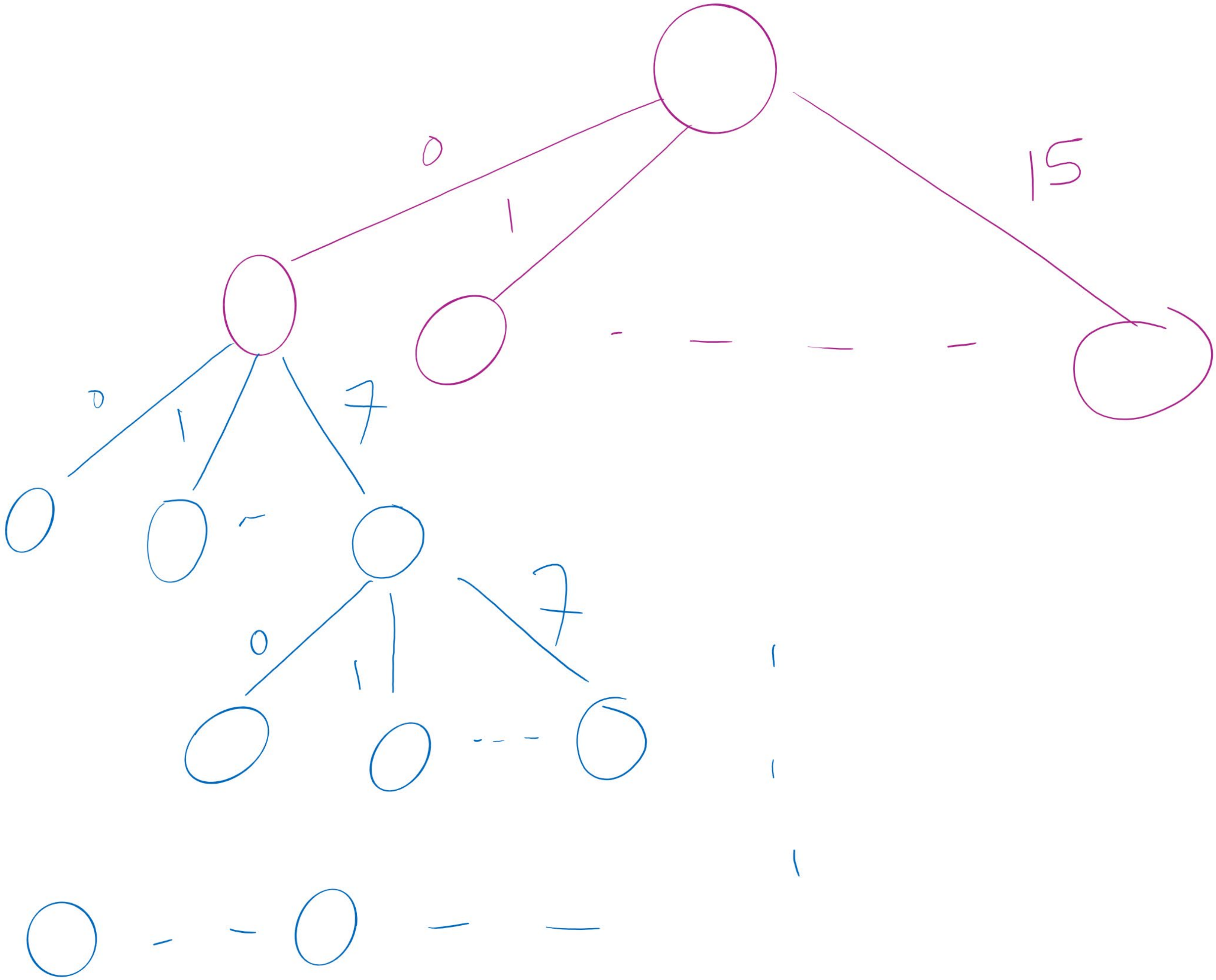
Another problem: Boggle

- Words at least 3 adjacent letters long must be assembled from a 4x4 grid
- Adjacent letters are horizontally, vertically, or diagonally neighboring
- Any cube in the grid can only be used once per word

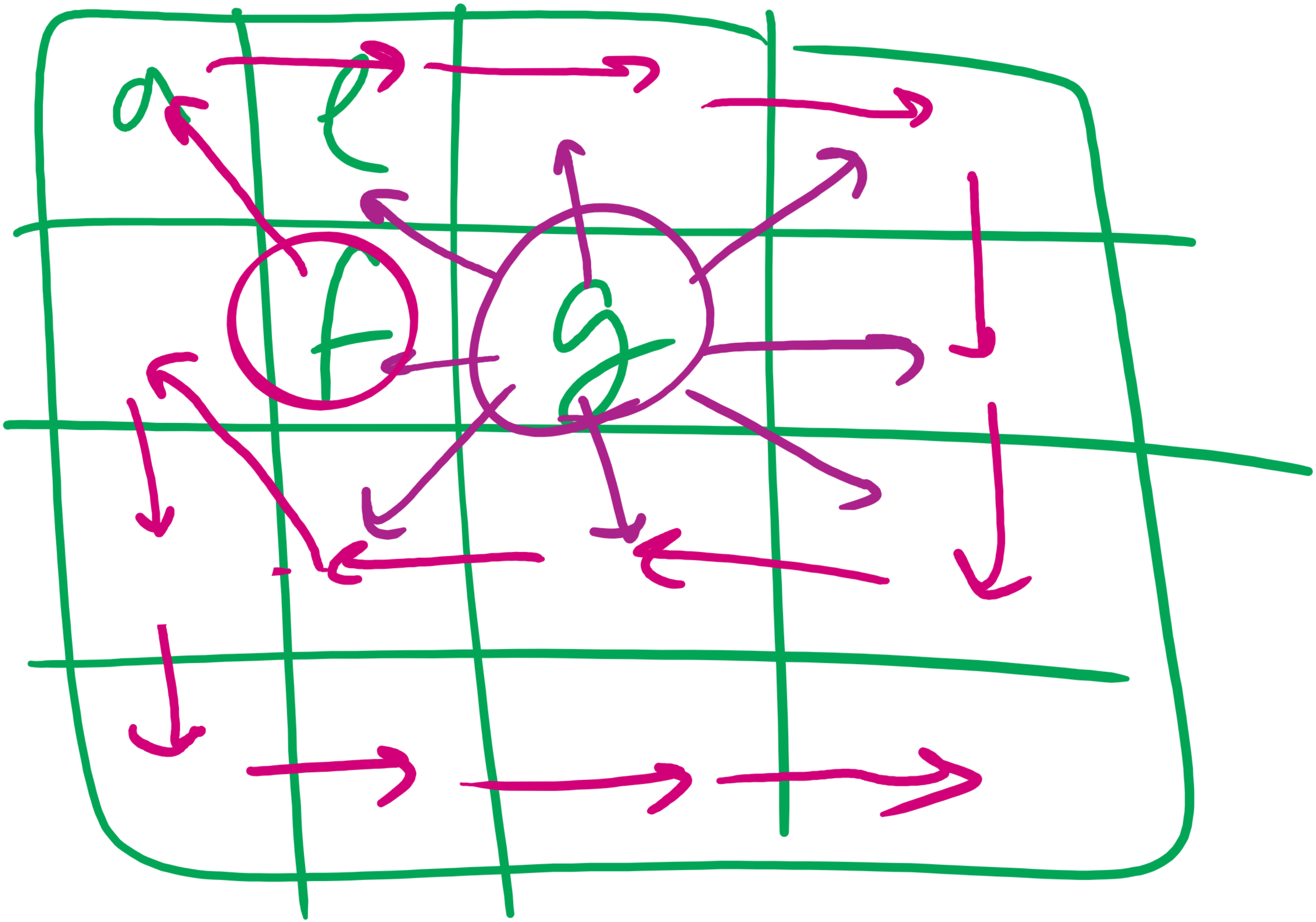
Recursing through Boggle letters

- Have 16 different options to start from
- Have 8 different options from each cube
    - From B[i][j]:
        - B[i-1][j-1]
        - B[i-1][j]
        - B[i-1][j+1]
        - B[i][j-1]
        - B[i][j+1]
        - B[i+1][j-1]
        - B[i+1][j]
        - B[i+1][j+1]

Search Space



Boggle Question



Are all 8 options valid?

- Can’t go past the edge of the board
- Can’t reuse the same cells in the board
- Each letter added must lead to the prefix of an actual word
  - Check the dictionary as we add each letter, if there is no word with the currently constructed string as a <u>prefix</u>, don’t go further down this way
  - Practically, this can be used for huge savings
- This is called pruning!

Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com